Recall: values and equality in Java

**Primitive types vs objects**

There are two types of values in Java: **primitive values** and **objects**.

Examples of primitive value types include `int`, `double`, `boolean`, and other builtin types. Java directly stores primitive values (it’s “in the box”).

Examples of object types include `String`, `LinkedList`, other library types, and user-defined classes. Java stores **references** to objects.

**Reference equality (==) vs value equality (.equals)**

Reference equality checks whether two names refer to the same object: `x == y`.

Value equality calls a method that (typically) checks whether two potentially different objects have the same value: `x.equals(y)`.

Java Conventions and good programming practices

These conventions and programming practices are not required or checked by Java, and not all of them are they universally agreed upon by all Java programmers. For consistency, though, we’ll follow them in CS 42.

- Place field definitions at the top of the class.
- Use Javadoc `/** ... */` to document your fields and methods.
- Inside a class, always use `this` to refer to the members (i.e., fields and methods) of the class.
- Keep your main program separate from your class definitions.
- Usually, fields are private, and the class provides public getters and setters, if needed.
- Write tests first!
- Minimize the number of methods that access fields.
- Write a `toString` method for every class you implement.
- Provide (preferably through autogeneration) an `equals` and `hashCode` method for every class you implement.
- Always refer to a static field via the class. Never refer to a static field via an object.
- Provide good constructors. Explicitly initialize all fields.

Encapsulation mechanisms are a social construct

An encapsulation mechanism is how a language distinguishes between the interface of an object (i.e., what it can do) and the implementation details of that object (i.e., how the object does its thing).

Each OOP programming language has its own encapsulation mechanisms (e.g., Java has `public` and `private`), and each differs in how strictly the language enforces the mechanism.

Encapsulation is **not** about security, and ultimately it’s up to people (not computers) to enforce it.
public class DrinkContainer {
    /**
     * the color of the container */
    private String color;
    /** how much liquid the container can hold, in mL */
    private int capacity;
    /** how full the container is, in mL */
    private int fullness;

    public DrinkContainer(String containerColor, int capacity) {
        this.color = containerColor;
        this.capacity = capacity;
        this.fullness = 0;
    }

    public String getColor() {
        return this.color;
    }

    public int getCapacity() {
        return this.capacity;
    }

    public int getFullness() {
        return this.fullness;
    }

    /**
     * Sets a new fullness for the container.
     * If new fullness is less than 0, then sets fullness to 0
     * If new fullness is greater than capacity, sets fullness to capacity.
     *
     * @param newFullness the new fullness of the container
     */
    public void setFullness(int newFullness) {

    }

    /**
     * Fills the cup to its capacity
     */
    public void fill() {

    }
}

Next time: sorting and summations